

UNCLASSIFIED

DRES



DEFENCE RESEARCH ESTABLISHMENT SUFFIELD

SR 709

UNCLASSIFIED

Techniques for Pattern Classification Using a Convergent Recurrent Neural Network

By:

Simon A. Barton

December 1998

WARNING

"The use of this information is permitted subject to recognition of proprietary and patent rights."



CRAD



National Defence Défense nationale

Canada

UNCLASSIFIED

UNCLASSIFIED

DEFENCE RESEARCH ESTABLISHMENT SUFFIELD
RALSTON, ALBERTA

SUFFIELD REPORT NO. 709

Techniques for Pattern Classification
Using a Convergent Recurrent Neural Network

by

Simon A. Barton

WARNING

"The use of this information is permitted subject to
recognition of proprietary and patent rights".

UNCLASSIFIED

102-02815/

UNCLASSIFIED

ABSTRACT

A recurrent neural network (RNN) using a structure and learning rules that have previously been shown to result in autonomous goal-driven motion in a simulated mobile machine, is now shown to be capable of memory, recognition and identification of patterns. Two techniques are presented. The first uses implicit memory, in which no direct representation of an image is stored, and an image may be identified by the point to which the RNN converges during exposure to the image. Data compression may be achieved by projecting the output of all the nodes in the RNN onto a vector that is of lower dimension than the image. This vector can be used to identify trained patterns.

It is also possible to store and regenerate explicit representations of the actual images used in training. To achieve this, the learning rules used in the first technique were augmented by a resonant feedback signal that adjusts the input connection strengths of nodes in a memory region of the RNN. An image that has been used in training then stimulates reproduction of the same image in an output array coupled to the RNN memory region, and generates a signal indicating recognition. An image that has not been seen before, generates no reproduction or recognition signal.

Based on the technique for explicit memory, designs for a self-learning associative memory are presented. These designs aim to create a system that is capable of learning to make connections between two different forms of sensor input to an RNN. By repetition of image pairs, this system would be able to perform such tasks as learning to name objects, classify images, or provide adaptive control for an autonomous vehicle.

UNCLASSIFIED

iv

UNCLASSIFIED

DRES-SR-709

UNCLASSIFIED

EXECUTIVE SUMMARY

The automatic recognition and identification of patterns in images is a fundamental requirement for advanced autonomous systems that are to be used for such tasks as target recognition in IR images, land mine detection, and the control of unmanned vehicles.

A recurrent neural network (RNN) is a collection of simple processing units (nodes or neurons) with feedback connections. It is well established that RNNs have some capability for self-organized image storage and classification, and previous work at DRES has shown that an RNN with input from sensors can control goal-driven motion in a simulated mobile machine.

RNNs therefore appear to have the potential to improve autonomous systems, by using unsupervised learning and adaptive control. RNNs have internal learning mechanisms, include temporal feedback of sensory information, are readily scalable, and offer the possibility of massively parallel computation.

In this report, the C++ software structure, implementation and user interface of a general RNN system is first described. This interface has been used to determine the capabilities of the RNN for remembering and recognizing patterns.

Two forms of memory are shown to be possible. In *implicit memory*, no direct representation of an image is stored, but images are identifiable by the point to which the RNN converges during exposure to the image. A form of data compression may be used, since the output of all the nodes in the RNN can be projected onto a vector that is of much lower dimension than the image, and this vector retains sufficient information to identify trained patterns. We call the implicit memory training method *modified Hebbian* (MH), as it is based on a biologically inspired mechanism proposed by Donald Hebb.

In *explicit memory*, the second form, close reproductions of the actual images used in training are possible. To achieve this, the MH method was augmented by *resonant feedback* learning in a memory region of the RNN. In this technique, memory is distributed among the input connection strengths of nodes in the memory region. When an image that has been previously used in training is presented to the RNN, through sensor nodes, it stimulates reproduction of the image in an output array coupled to the RNN memory region, and at the same time generates a signal indicating recognition. If an image is presented that has not been seen before, no reproduction or recognition signal is generated.

Both memory methods are self-organizing and the potential for automatic feature extraction is discussed. However, as yet neither method has an internal method for *association*; i.e. they can not yet *learn to name* objects or patterns by repeatedly being

UNCLASSIFIED

told what they are. The RNN structure does not preclude this capability, however, and two possible designs that may lead to self-organized associative memory are presented.

The designs for a self-learning, associative memory are based on the technique for explicit memory. These designs aim to create a system that is capable of learning to make connections between two different forms of sensor input to an RNN. By repetition of image pairs, this system would be able to perform such tasks as learning to name objects or classify images. Future work will implement and test these designs, and will attempt to link automatic associative memory with goal-driven motion control to create a simulation of an adaptive, autonomous machine that can learn to improve its performance in obstacle avoidance and navigation.

This would be highly desirable for an autonomous vehicle operating in an unknown and dynamically changing environment, and a system that has a mechanism for automatic, self-organized extraction of important features, with relevance based on the associated context would have great potential for creating a general solution to the pattern recognition problem. It is possible that recurrent neural networks have the necessary properties to enable us to achieve this goal if we can build suitable architectures and learning rules. This report is concerned with one step towards that goal; namely demonstrating that RNNs do have the capability for self-organized memory, recognition and classification of images.

This work was performed under project 2FE11, "Advanced Adaptive Control Theory and Applications", and will continue partly under that project and partly under a three year Technology Investment Fund project, 2FE20, "Self-Organized, Goal-Driven Adaptive Learning", beginning April 1999.

Table of Contents

Abstract	iii
Executive Summary	v
List of Figures	ix
List of Tables	xi
1 Introduction	1
2 RNN Structure	7
2.1 Node Signal Processing	8
2.2 Information Storage	9
2.3 C++ Implementation and Interface	10
3 Implicit Memory (Modified Hebbian Learning)	17
3.1 MH Training and Classification Procedures	18
3.2 MH Results	19
3.3 Problems with the MH Technique	22
4 Explicit Memory (Resonant Feedback Learning)	25
4.1 RF Results	28
5 Associative Memory Design	31

UNCLASSIFIED

5.1 Cross-Linked Structure	32
5.2 Circular Structure	33
6 Discussion and Conclusions	35
References	39

List of Figures

2.1	RNN with 24 sensor, 100 recurrent and 6 motor nodes	8
2.2	The C++ MakeRnn dialog	11
2.3	The RnnPr interface for using an RNN	14
2.4	The RnnPr Parameters dialog	15
3.1	Ten training images using 24 sensors (4x6 array)	19
3.2	Two examples of RNN output after MH training	20
3.3	Five training images using 256 sensors (16x16 array)	22
4.1	An MH-RNN with Resonant Feedback	26
4.2	Total correlation of ten 4x6 images during RF training with $\rho = 0.02$ (smooth) and $\rho = 0.10$ (unstable)	29
4.3	Two examples of RNN output after RF training	30
5.1	RF-RNN cross-linked associative memory design	32
5.2	RF-RNN circular associative memory design	33

UNCLASSIFIED

x
UNCLASSIFIED

DRES-SR-709

List of Tables

I	Effect of increased noise and number of Mnodes on RNN identification of 4x6 images after MH training with fixed learning parameters . . .	21
II	Optimal RNN identification of 4x6 images with 40% maximum noise, after MH training	22
III	RNN identification of 16x16 images after MH training	23

1. Introduction

Recognizing patterns in data is a necessary component of all scientific activity. The scientific method is a cycle of two parts, experiment and hypothesis, which is constantly iterated to generate a self-consistent theory. It is in the analysis of experimental results, their relation to the theory, and the development of new or refined hypotheses, that pattern recognition is essential. Indeed, it is arguable that most of the mental activity of intelligent animals is devoted to the recognition and association of patterns.

Attempts to develop an intelligent machine are hampered from the outset by the difficulty in defining what we mean by *intelligence*. However, it is clear that the ability to recognize patterns must be included as one of the fundamental properties of any intelligent machine. Other necessary properties include the ability to *interact* with the environment; i.e. it must receive information from sensors, and it must be able to activate motor responses that cause some change in its environment. Furthermore, the decision mechanism for action must be *adaptive* and must be related to the achievement of a goal. Of course, defining intelligent behaviour as including pattern recognition leads to the substitution of one property that is hard to define with another. What do we mean by *pattern recognition*? What is recognition? Can we recognize something that has no pattern?

Human beings can sense many kinds of pattern, in space and in time, or both together. A pattern may be large; we can recognize our entire field of view as being, for example, "the city of Calgary viewed from the air", or we can focus on regions of our view and describe individual objects. We can also hear a sequence of sounds over a short or long time and recognize a tune, an entire symphony, the American Declaration of Independence, or a single word, which we relate to an object, or an idea, a smell, a picture, and so on. We are also able to use the fact that we recognize relationships between sets of data in order to propose a theory about how the world and the universe operate, or the nature of existence as it impinges on our senses. Patterns and their relationships pervade and define our existence, but the concept of pattern recognition is not easy to define.

Recognition will be taken to mean an *indication of familiarity*; i.e. if I recognize something, I can say "I have seen this before", without necessarily knowing what it is, or what its properties are.

Detection of a pattern is related to recognition, because in order to detect that a pattern exists in data you must be able to say "I've seen this before", even if the repeated part is only in an image that you are seeing for the first time.

Identification, which will be equated to classification, is the ability to name a recognized pattern. For this to be done autonomously, it requires an internal mechanism for *association*, which is the linking of two kinds of pattern derived from separate sensor sources relating to the same experience; e.g. imagery from sound and vision sensors may be stored in such a way that words are linked to objects.

Memory is required for these processes. We will use the term *explicit memory* to mean storage of sensor data in such a way that it can be recalled as it appeared, more or less, at the sensor during the memorization process. This does not mean that every memorized image is necessarily stored pixel by pixel, but that a close reproduction of a memorized image can be generated by the action of some stimulus (sensory input). This could also be called direct memory.

By *implicit memory* we mean storage of some *representation* of the sensor data; e.g. compressed or as features. This is indirect memory. In this case, the original image can not be reproduced directly.

Recognition itself implies a pattern, for there must be an indication of matching between stored and sensor images for recognition to occur. Of course, it is possible to identify random noise, which is the absence of pattern. However, the absence of pattern can be identified because there is no region that is recognizable; i.e. we call the absence of pattern "random", and that is an identification not a recognition.

In this work then, we equate classification with *identification*, rather than with recognition. This may seem a subtle and unimportant distinction, but the result is that a *pattern classifier* does not require a mechanism for explicit memory, whereas internal reproduction of an image does. It may be that autonomous association also requires explicit memory.

The use of different sensor data streams to arrive at a conclusion or determine an action, is an example of *data fusion*, and is of great use in many applications, particularly those that employ machines to act autonomously. Data fusion may be achieved by a mechanism of association.

While accepting that pattern recognition may be applied to time sequences and multisensor data fusion, this work focuses on the recognition and identification of objects in spatial imagery, such as IR or visible camera images. This is relevant to several areas of ongoing R&D at DRES, including the detection and identification of: land mines and targets in IR images; objects in visible (video) images for surveillance, obstacle avoidance and navigation in unmanned ground vehicles; and a Technology Investment Fund (TIF) project (2FE20) with the long term goal of demonstrating self-organized, goal-driven adaptive learning in a simulated mobile vehicle with multiple sensors.

We have seen that pattern recognition is difficult to define. It is also impossible to implement in a general way with present techniques. Using the standard current computer methods to analyse data from sensors, pattern recognition reduces to a *classification* technique. An array or vector of values is presented as input to an analysis system. This generates an output that either identifies the input as belonging to the nearest known class, or indicates that the input is not near enough to any known class to be identifiable. In this case, some systems are capable of creating a new class. Clearly, in this approach recognition becomes classification. The determination of which class is the nearest may be made on a *decision theoretical* basis or on a *syntactical* basis [1], but a predetermined value for some decision parameter is required, and this is also used to determine when an input is to be labelled unknown. Also, how the classes are to be *identified* (or named) must be decided in advance by the human designer, because there is no internal mechanism to associate names with classes.

The standard mathematical procedure in pattern recognition is to reduce sensor data to a classification using a three step process [1]: acquisition, preprocessing, and classification. In the data acquisition stage, a digital image is generated and will not be discussed further. The preprocessing and classification stages comprise the recognition process in the standard technique. Preprocessing normally involves some form of image enhancement (e.g. contrast normalization, edge enhancement, shadow removal), followed by feature extraction. During preprocessing of a large image, such as a 640 x 480 frame, it may also be necessary to have a technique to find smaller regions, say 32 x 32 pixels, in which it is most likely that recognizable objects will occur; this is sometimes termed *fixation*, and can greatly reduce the computation time. A feature vector is the normal output of the preprocessor. The classification stage then takes the feature vector and generates an identification.

In this approach, it is not necessary for the classifier to have any explicit memory of the patterns that it has been programmed to identify. Identification may be based on the image *properties* or *features*, and not on the images themselves. The perceived advantage of this is that it may be possible to define *invariant* features; i.e. features that do not depend on the size of an object in the image (the scale), or on its orientation (rotation and angle of view). Though this is may be possible for specific objects under well defined conditions for some images, it is often extremely difficult to implement in practice, and may be impossible to generalize. Furthermore, an adaptive system must be able to add new classes when new objects are to be identified, and this may mean that the chosen feature vectors become inappropriate for reliable identification, and then the rules for classification must themselves be adaptive. Each pattern classification application must be optimized individually, based on the ability of human experts to reliably and efficiently define, extract, and correctly group features of an image. Problems of scale, translation, rotation, and orientation of view may be extremely difficult or impossible to overcome.

A more flexible technique, that is self-organizing and adaptive, with the potential

for explicit memory and association, is desirable.

It is well established that neural networks with recurrent (feedback) connections have some capability for self-organized image storage and classification [2]-[6]. This report is concerned with the potential for memory, recognition and identification in a recurrent neural network (RNN) based on the structures and learning rules that have previously been shown to result in goal-driven behaviour in a simulated mobile machine [7],[8]. In that simulation, a mobile machine with light sensors was able to follow a moving light source, even in the presence of obstacles, which it was able to detect with sonar sensors and avoid using a separate reflexive mechanism. The motivation for this work is not only to develop a general system for pattern memory, recognition and classification (for possible use in mine detection and automatic target recognition applications), but also to demonstrate that an RNN system has the basic properties required for the development of an autonomous, intelligent mobile machine. The long-term aim is to link goal-driven behaviour with memory and association of multi-sensor input to provide a demonstration of adaptive, self-organized learning to improve the control (navigation and obstacle avoidance) of the simulated machine.

A recurrent neural network is a set of interconnected, simple processing units, called nodes or neurons. Chapter 2 describes the RNN structure employed in this work. Sensor nodes provide information to the RNN, and signals are passed between the internal nodes by connections of varying strengths (called weights). Each node has many input connections from other nodes, and as the RNN operates, the connection weights may vary with the signal strengths of the connected nodes. This can lead to memory and adaptive control capabilities, by pattern storage and response to sensors, whose signals complete a feedback loop with the environment. The RNNs used here are nonlinear feedback systems that can operate in stable, periodic or chaotic regimes [8]. By stable, we mean that if left with a constant pattern on the sensor nodes, the RNN signals eventually converge to fixed values. The transition from stability to periodicity and chaos can be controlled by the choice of value ranges for parameters that control the node responses and the learning rates. In previous work [9] we have established conditions under which the RNNs will be stable, and methods for verifying stability. The present work uses RNNs operating in the stable regime.

In Chapter 3, we describe a method for generating implicit memory of patterns in an RNN. This method is a form of *clustering*, because different patterns that are repeatedly presented to the RNN drive the system to different fixed points, a unique one for each pattern. Classification is then possible if the user records which fixed point corresponds to each class to be identified. It will be shown that a type of data compression is possible with this method.

The second method for storing patterns is described in Chapter 4. This method generates explicit memory of sensor images that are presented repeatedly to an RNN. The method uses a feedback signal derived from the differences between sensor and memory output nodes, mapped in a 1:1 way, to adjust connection weights in the

memory region of the RNN:

Both methods are self-organizing and the potential for automatic feature extraction is discussed. However, as yet neither method has an internal method for association; i.e. they can not yet *learn to name* objects or patterns by repeatedly being told what they are. The RNN structure does not preclude this capability, however, and two possible designs that may lead to self-organized associative memory are presented in Chapter 5. The implementation and testing of these structures is ongoing.

The ideal, general pattern recognition system would be *adaptive and autonomous*, so that it would not rely on a limited number of preprogrammed classes and their decision criteria based on the knowledge of a human expert, as in the standard techniques. The ideal system would also be independently able to add to its known set of classes (i.e. expand its memory and associations). This would be highly desirable for an adaptive, autonomous vehicle operating in an unknown and dynamically changing environment, and a system that has a mechanism for automatic, self-organized extraction of important features, with relevance based on the *associated context* would have great potential for creating a general solution to the pattern recognition problem. It is possible that recurrent neural networks have the necessary properties to enable us to achieve this goal if we can build suitable architectures and learning rules. This report is concerned with one step towards that goal; namely demonstrating that RNNs do have the capability for self-organized memory, recognition and classification of images.

UNCLASSIFIED

UNCLASSIFIED

... DRES-SR-709

2. RNN Structure

The basic structure and properties of the recurrent neural network used in this work have been described previously [9]. To provide greater structural flexibility, as well as a better graphical environment and user interface, the original C code implementation has been rewritten with the object-oriented programming (OOP) philosophy using Microsoft Visual C++. In this approach, an RNN is a class object containing arrays of nodes that are also written as class objects (the `NodeArray` class). Sensor, recurrent and motor node arrays constitute an RNN, and their interactions are defined by the class functions (methods in OOP terminology). Sensor arrays provide the RNN input, and motor arrays the output, which in general could drive the actions of a machine, or provide memory recall or classification. With this approach it is relatively easy to adjust the RNN architecture by adding more sensor or motor arrays, increasing the recurrent node array size, and defining the regions of the RNN that interact with sensor and motor node arrays. Interactions between separate RNNs are also possible. It is also feasible to split the computation of the RNN activity into multiple execution threads that can then be executed in parallel on a multiprocessor computer under the Windows NT operating system.

An example of an RNN structure is shown in Fig. 2.1. For brevity, the three types of node will be referred to as Snodes (sensor), Rnodes (recurrent) and Mnodes (motor). In Fig. 2.1, there are 100 Rnodes, and 24 Snodes transfer information to an input region of the Rnodes. The size of the input region is definable, up to the total number of Rnodes. Six Mnodes are also connected to an output region of the Rnodes whose size is similarly definable. For the network shown, the input and output regions do not overlap, and are shown in different shades of grey. In this example, each region uses half of the 100 Rnodes. The figure shows a 4x6 sensor array containing a simple pattern, the number one, and some grey-scale representation of this pattern, as generated by the RNN and displayed in the 6 Mnodes.

Signals are transferred between the nodes by connections, each of which has an associated weight. Each Snode has a specified number of output connections, and each Rnode and Mnode has a specified number of input connections. Once a number of connections is specified, they are established randomly. Snode connections are chosen from Rnodes in the input region, and Mnode connections are chosen from Rnodes in the output region. Establishing Rnode input connections is slightly more

UNCLASSIFIED

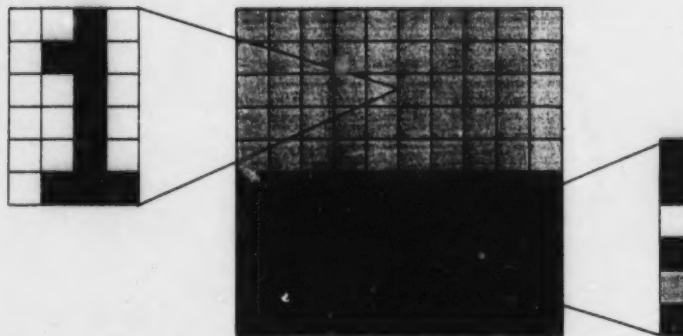


Figure 2.1: RNN with 24 sensor, 100 recurrent and 6 motor nodes

complicated, because the Rnodes in the input region already have some inputs from the Snodes, so these are counted before finding inputs from other Rnodes. All the Rnodes can receive input from any other Rnode, chosen randomly. Self-connection is not allowed in the current RNN implementation, but its inclusion would present no difficulty.

To prevent the possibility of disjoint sets of Rnodes (having no connection outside the group), the first input connection to each Rnode is always from the previous Rnode in the array. To complete the loop, the first Rnode in the array receives input from the last, and all other inputs are then established randomly.

The set of connections for each type of node is forced to be unique. No Snode can have more than one output to any Rnode, and no two Snodes can have the same set of outputs. No Mnode can have more than one input from any Rnode, and no two Mnodes can have the same set of inputs. No Rnode can have more than one input from any other node, and it is guaranteed that two Rnodes cannot have the same set of inputs because of the loop structure and the exclusion of self-connection. The loop structure also ensures that every Rnode has at least one output, and in practice the random process ensures that every Rnode has about as many outputs as inputs.

For a given number of Snodes, Rnodes and Mnodes, reference [9] discusses the limitations on the number of input and output connections for each type of node. It is important that since each Snode is connected to a unique set of Rnodes, and each Rnode has a unique set of input connections, the input region of the Rnode array generates unique combinations of the input patterns at each Rnode.

2.1 Node Signal Processing

Each Rnode receives signals from many other nodes. The connection weights are used to form a weighted sum of the input signals, and this total weighted input is processed by a nonlinear function of the *sigmoidal* form. This is characterized by an exponent

UNCLASSIFIED

DRES-SR-709

s , which determines the steepness of the function, and an offset x_0 , which fixes the centre of its output range. The total weighted input to the n th node at cycle t is:

$$x_n^t = \sum_i^{n_{icons}} W_{ni}^{t-1} f_j^{t-1} \quad (2.1)$$

where n_{icons} is the number of input connections, and the subscript j gives an index for the node providing input on the i th connection, with weight W_{ni}^{t-1} . The input connection weights are stored in a matrix W of dimension (number of Rnodes) $\times$ (n_{icons}), and the input connection indices are stored in a matrix C of the same dimension. Thus:

$$j = C_{ni} \quad (2.2)$$

In the C++ implementation, the NodeArray class objects also store output connections and weights, which may be useful in implementing feedback and feedforward learning methods, as discussed in Chapter 4.

The sigmoidal form of the node output is:

$$f_n^t = [1 + \exp(-s(x_n^t - x_{0n}^{t-1}))]^{-1} \quad (2.3)$$

The steepness of the sigmoid curve increases with s , which could be different for each node. In this RNN model, the same fixed value is applied to every Rnode (s_r). Another fixed value (s_o) is used for each Mnode. The NodeArray class does, however, store a vector of s values, one for each node, so it would be easy to implement different values for each node. The Snodes simply provide values between 0.0 and 1.0 from some external source and do not have any sigmoidal processing.

2.2 Information Storage

Biological brains are vast arrays of neurons with massive recurrent (feedback) connectivity. The ability of the neural connections to change their strengths is believed to allow an animal to remember and learn from its experience [10]. A machine using an RNN whose connection weights and offsets can change from cycle to cycle may therefore be expected to have the potential for memory and learning. The criteria used for changing the variable parameters in a neural network are therefore called the *learning rules*. Using different learning rules will lead to different properties in an RNN.

Allowing the Rnode weights and offsets to change can provide a mechanism for repeated input patterns to affect the internal processing and thereby affect the output of the network. For pattern recognition, the goal may be to find a learning rule that

causes repeatedly seen patterns to generate particular strong and weak connections between Rnodes, so that when seen again a pattern will generate a reproducible and distinct output in the Mnodes, which can then be used as a classification. This is the approach followed in the learning method described in Chapter 3. On the other hand, for explicit memory and recognition of patterns, the aim is to create a connection weight and node offset structure that contains the information necessary for the RNN to reproduce, in the Mnodes, a pattern that it has seen repeatedly, and this is the learning method described in Chapter 4.

For motor control, the Mnodes provide signals that cause changes in the position of a machine, and hence a change in the input to the sensors. Thus, feedback from the environment of repeated experience may generate goal-driven behaviour [7] [8], if an appropriate learning rule is used.

Many learning rules have been proposed in the literature, most of which attempt to generate stable network states in response to given initial conditions. This is not necessarily appropriate for motor control, and it is not yet certain that it is required for memory, recognition or association. References [11] and [12] are collections that contain most of the important early papers on learning algorithms.

2.3 C++ Implementation and Interface

In this C++ implementation of an RNN, it is first necessary to create an RNN structure and initial state. This is achieved by running the initialization program named **MakeRnn**, which executes as a dialog based application; i.e. a dialog box appears with default values for the number of nodes and connections, and parameters for the generation of random weights and offsets. Figure 2.2 shows the dialog box with the default values for the parameters, each of which may be edited by the user.

The initialization program **MakeRnn** establishes a network structure and state. The state is completely defined by the following: the number of Snodes, Rnodes and Mnodes; the number of input connections for each type of node; the connection index and weight matrices; the node outputs; and the node offsets. When using the RNN (see below), each type of node is displayed as a 2D array on the computer screen, so it is necessary to specify the number of nodes as x and y dimensions, the total number being the product xy . To define the RNN structure it is also necessary to specify the regions of the Rnodes that can have input from the Snodes, and that can provide output to the Mnodes. These regions are specified in the Minimum and Maximum Rnode connection boxes for the Sensor and Motor nodes.

The connection weights may be positive or negative, corresponding to excitatory or inhibitory synapses (connections) in biological neural systems. A biological neuron may supply an excitatory signal to one connected neuron, and an inhibitory signal to another [13]. This scheme has been maintained in the RNN model; i.e. the node

outputs are all between 0 and 1, but the weights connecting a neuron to others may be positive or negative.

The weights for connections from Snodes to Rnodes, and from Rnodes to Mnodes are kept constant, because these connections simply transfer information. Any learning or adaptation must occur in the body of the RNN; i.e. learning occurs within the Rnodes.

The magnitudes of the Rnode connection weights are random, between the limits supplied by the user. The defaults are 0.1 to 0.9, as shown in Fig. 2.2. A fraction of positive Rnode-Rnode weights is also supplied by the user, and the program assigns weight signs randomly using the desired fraction as a weighting factor. Thus a user may generate an RNN that is predominantly excitatory or inhibitory.

The Snode to Rnode weights are all initially set to 1.0, and the Rnode to Mnode weights are also 1.0, but have alternating signs (+1, -1, +1, -1,...).

The Rnode offsets are all initialized randomly between user defined limits (defaults -1 to 1 in Fig. 2.2). The Mnode offsets are set to zero. Thus, the Mnodes with fixed weights of alternating sign initially have input signals close to zero on average.

Output signals for the Snodes and Mnodes are all initially set to 0.0, and for the Rnodes, they are set randomly between 0.0 and 1.0.

Learning parameters that affect the rates of change of the Rnode connection weights and offsets may also be initialized at the bottom of the dialog box. These are explained in Chapter 3. The output precision sets the number of significant figure to the right of the decimal point that are printed to a file when the RNN construction is finished.

MakeRnn generates a uniquely named output file of the form **rnnxy.stt**, where **xy** is an integer, which contains all the structural and state information using the parameters supplied by the user.

After generating the initial state file, an RNN is set into iterative operation by executing the Windows program **RnnPr**.

Figure 2.3 shows an example of the graphical interface. Initially the user is presented with a blank view, and must open an RNN state file generated by **MakeRnn**. The user is then prompted to open an image file, a file containing patterns that the RNN will be exposed to. The image file is a sequence of bytes that are interpreted as integers between 0 and 255 and converted to floating point values between 0.0 and 1.0, to be used as the Snode output signals. The user is prompted to confirm the number of patterns in the image file, and the first pattern is displayed in the Snode array. A simple representation of the number zero is shown in Fig. 2.3. Initial values for the Rnode and Mnode signals are also shown as color representations of values in the range 0.0 to 1.0, but have been converted to a grey scale for the display in this report. The Color menu item allows the user to display the Mnode signals as a grey scale, which may be more useful if the RNN is being used for explicit memory

(Chapter 4).

It is necessary to click the **Parameters** menu in order to set initial and final numbers of RNN cycles, and to adjust any of the other defaults in the **Parameters** dialog box, shown in Figure 2.4. The **Exposure time** is the number of RNN cycles for which each pattern will be retained in the **Snode** array before moving to the next pattern in the image file. The **Intensity factor**, **Background** and **Maximum noise** allow the user to adjust the pattern properties. For example, a simple pattern of zeros and ones may be adjusted to have a background level of 0.4 and a maximum signal above that of up to 0.6 (the sum can not exceed 1.0), and noise may be added to every pattern element, again such that the signals lie between 0.0 and 1.0.

The **Node** parameters allow the user to adjust the sigmoidal steepness values, and the constant weight between **Snodes** and **Rnodes** can also be varied. The **Learning** parameters affect the rates of change of the **Rnode** connection weights and offsets, and the resonant feedback rate (**RF**) and width are explained in Chapter 4. The output precision sets the number of significant figures to the right of the decimal point that are output if the state is saved, using the **Save** menu item.

After clicking on the **Start** menu item, **RnnPr** iteratively calculates the RNN node signals, adjusting the weights and offsets after each iteration for the designated number of initial cycles. During this time the graphical display of the node signals is not updated, to minimize the execution time.

After an initial number of cycles, which may be chosen to be zero, **RnnPr** executes a final number of cycles, which again may be zero, during which the graphical interface is updated after each cycle. The user can then see the activity of the RNN as different patterns are presented to it, and the unique **Mnode** output arrays that may be generated.

The entire RNN state may be reset to the initial conditions by clicking the **Re-Initialize** menu item, and the **Refresh** item updates the graphical display.

The **Learning** menu item presents a choice of Modified Hebbian or Resonant Feedback learning methods, which are explained in Chapters 3 and 4 respectively.

With this graphical interface it is very easy to study the pattern recognition and classification properties of an RNN, and to see the effects of varying the structural and learning parameters and methods. It also provides a flexible environment that can be easily expanded to include new techniques and more complex RNN structures.

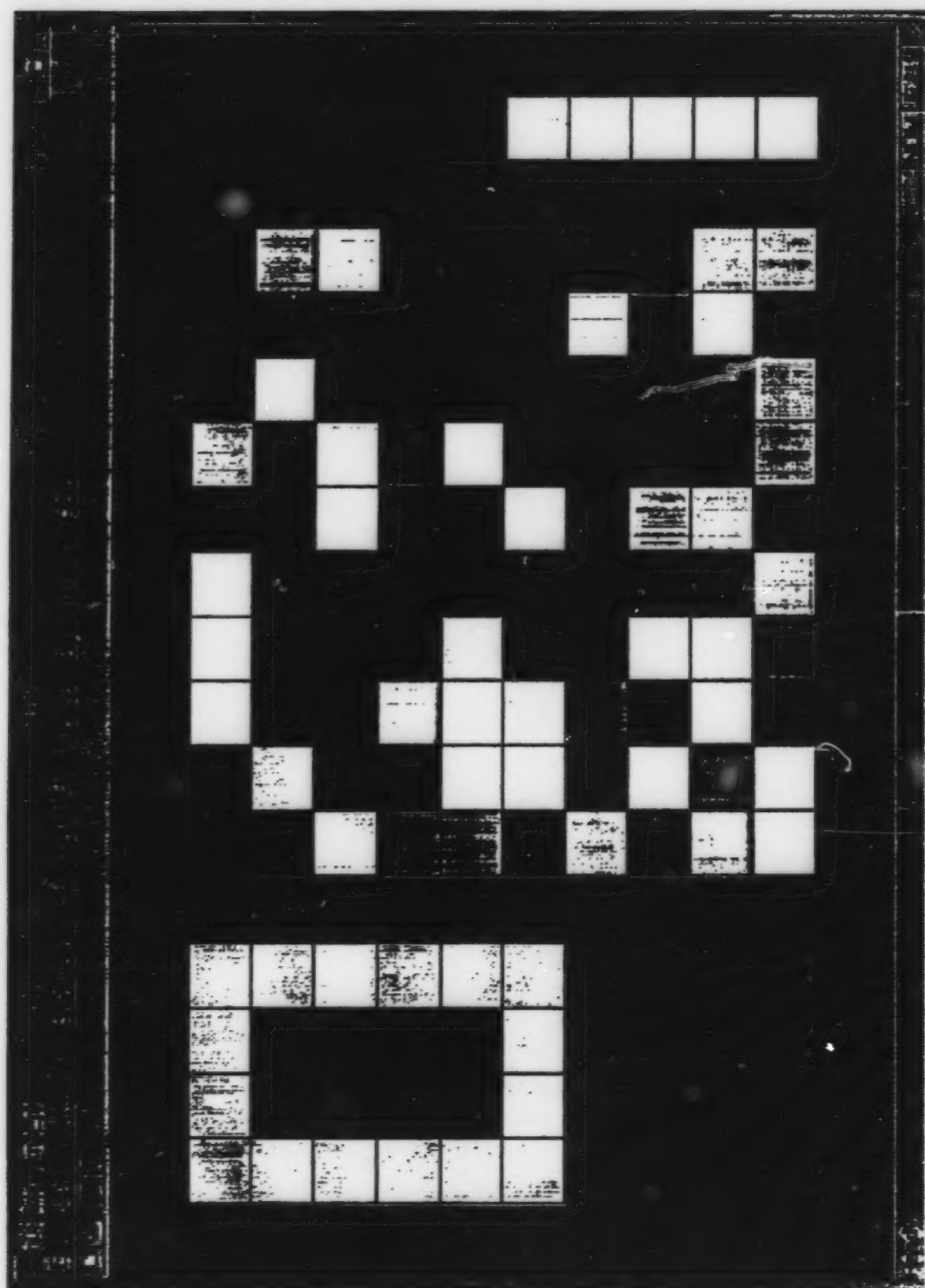
UNCLASSIFIED

Figure 2.3: The RnnPr interface for using an RNN

UNCLASSIFIED

DRES-SR-709

Parameters

Initial Value	0	Initial Value	10
Initial Value	0	Initial Value	0
Initial Value	0.4	Initial Value	0
Initial Value	0.4		
Initial Value	0.1	Initial Value	4
Initial Value	1		
Initial Value	0.001	Initial Value	0.0002
Initial Value	0.001	Initial Value	0
Initial Value	0.4		
Initial Value	8		

Figure 2.4: The RnnPr Parameters dialog

UNCLASSIFIED

UNCLASSIFIED

DRES-SR-709

3. Implicit Memory (Modified Hebbian Learning)

A modified form of the rule proposed by Hebb [14] for changes in the synaptic connection strengths in biological neurons was used in previous work [7] [8], where it was shown to result in goal-driven behaviour in an RNN whose output controlled the movement of a simulated machine. In this Chapter we show that the same modified Hebbian (MH) learning scheme can lead to a form of implicit memory of images that are repeatedly presented to an RNN.

The basic idea of Hebbian learning is that neural connections are strengthened when a strong input signal results in a strong output signal from the neuron receiving the input. Two modifications to that idea are used here. The first is to add a decay term to ensure that a weight cannot grow in strength without limit, and the second allows the node offsets to shift continuously towards the centre of the node output range. This second modification ensures that no node will continuously give output of 0.0 or 1.0, as it could if the offset were fixed at an inappropriate value. By allowing the offsets to change, all the nodes remain active, independent of the average signal level that an individual node may receive.

In this (MH) learning model, the Mnode offsets also change at every cycle in order to allow their output centres to shift to the average level of input that they are receiving.

Our MH learning then consists of two equations (3.1 and 3.2), which are executed at every network cycle. For an Rnode n at cycle t , which received an input f_j^{t-1} from the j th node and generated an output f_n^t , the change in the weight W_{ni} connecting nodes n and j is:

$$\Delta W_{ni}^t = \alpha S_{ni} f_j^{t-1} f_n^t - \gamma W_{ni}^t \quad (3.1)$$

where S_{ni} is the sign of the weight. Recall that the connection index j is given by C_{ni} (Eq. 2.2). Because the weights may be positive or negative, the sign term ensures that both types can increase in magnitude, and the decay term (γ) ensures that the weight magnitudes are bounded. The parameter α controls the maximum change that can occur at each cycle; i.e. it is a growth rate.

The change in the offsets of the Rnodes and Mnodes is given by:

$$\Delta x_{0n}^t = \beta(f_n^t - 0.5) \quad (3.2)$$

i.e. the offset of every node is adjusted so that the output approaches the centre of its range. In a convergent recurrent system with a fixed input image, all the node outputs eventually approach 0.5. For networks that receive changing sensor input, or are operating in a nonconvergent regime, the offsets change continually.

For a convergent system with fixed sensor input, the weight changes approach zero and Eq. 3.1 leads to the following:

$$|W_{ni}| \rightarrow \alpha/4\gamma \quad (3.3)$$

For example, if the growth rate is ten times the decay rate, then all the Rnode weights approach a magnitude of 2.5.

3.1 MH Training and Classification Procedures

During MH training, the following occurs:

- a set of different images is presented sequentially to the RNN through the sensor array;
- each image is held in the sensor array for a specified number of cycles, which we call the *exposure time*;
- the sequence of image presentations is repeated many times, until the Mnode outputs do not change significantly for any image; each image now generates a unique Mnode (output) vector;
- the Mnode vectors are then stored as *class vectors* to be used in identification.

During classification, the MH training is inactive, i.e. the weights and offsets are fixed at the values which existed at the end of the training cycles, and the following procedure is used:

- an image is retained in the sensor array for the exposure time;
- the Mnode vector that is then generated is compared with the stored class vectors, and the "closest" match identifies the input image.

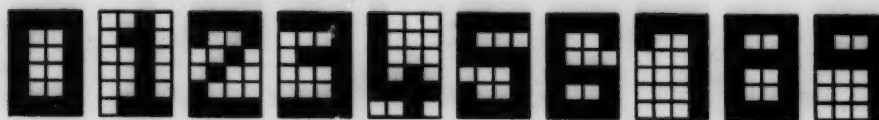


Figure 3.1: Ten training images using 24 sensors (4x6 array)

For n_m Mnodes, the n_m -dimensional distance (d_{n_m}) between two vectors (u, v) was used in this work as a measure of closeness. It is defined by:

$$d_{n_m} = \sqrt{\sum_i^{n_m} (u_i - v_i)^2} \quad (3.4)$$

3.2 MH Results

Any image that is presented to a convergent RNN using MH learning will eventually generate output signals in the Rnodes and Mnodes that are all equal to 0.5 because of the offset shift given by Eq. 3.2, providing that β is small enough to prevent oscillations. If there are n_r Rnodes, the RNN state moves on a trajectory in an n_r -dimensional space. However, it may take many thousands of RNN cycles to achieve convergence, and the trajectory followed is unique for every unique sensor image. Thus, if the exposure time is quite short, a unique and *quasi-stable* (very slowly changing) RNN state is achieved for every unique image. What is a short exposure time depends on the values used for the learning parameters α, β , and γ . For the conditions used in this work, exposure times in the range 10-100 cycles were used.

Since an RNN starts out in a random state, it is necessary to run the system for a number of initial cycles, which is set in an edit box of the **RnnPr Parameters** dialog (see Fig. 2.4). When a set of images is repeatedly presented to the RNN for the initial number of cycles, each image is held in the sensor array for the exposure time (in cycles). It should be noted that the total number of RNN cycles performed is therefore the exposure time multiplied by the number of initial cycles multiplied by the number of images. For example, if there are 10 images, with 100 initial cycles and an exposure time of 10, the total number of RNN cycles performed will be 10,000.

The initial training cycles serve to move the RNN into a state whose response is centred around the image set used, and is reproducible for each image. After the initial cycles, all the Rnodes respond rapidly (within a few cycles) to an input image, and produce an output in the Mnodes that is a unique representation of the image. The Mnodes in this technique can be viewed as a projection on n_m space of the n_r -dimensional space of the Rnodes. If n_m is less than the number of Snodes n_s containing the image, the Mnode vector is also a *compressed* representation of the image.

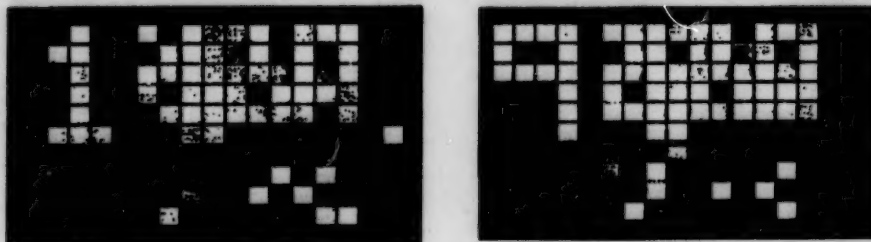


Figure 3.2: Two examples of RNN output after MH training

Figure 3.1 shows ten 4x6 images that were used as a test training set for an RNN consisting of 24 Snodes, 100 Rnodes, and a variable number of Mnodes (4, 6, 8, ...). In the test configuration, each Snode had 10 output connections, and the Rnodes and Mnodes each had 10 input connections. The constant weight between Snodes and Rnodes was set at 5.0, and the learning parameters were the following: $\alpha = 0.005$, $\gamma = 0.0005$, $\beta = 0.002$. The Rnode sigmoidal steepness parameter was $s_r = 0.1$, which ensures convergent behaviour for this RNN configuration [9].

Figure 3.2 shows a grey-scale version of two typical examples of the RNN output after MH training with the test image set. In the actual computer screen graphical display, to emphasize the unique node outputs, a color table is used which represents the signal values quantized in 0.025 steps between 0.4 and 0.7, where most of the nodes are normally active. Outside that range, grey levels are used. This color system generates visible representations of the RNN states that are readily seen to be quite different. Even with the grey-scale version of the color output shown in Fig. 3.2, the two different images clearly generate unique RNN states and Mnode vectors, as do all the test images.

The performance of an MH RNN as an image classifier can be assessed by studying its success in identifying noisy versions of the training images. Recall that one class vector for each trained image is stored after training, and identification is established by finding the class vector that is closest to the Mnode vector resulting from the presentation of a noisy image.

Table I shows identification results for an RNN using the parameters specified for the test configuration above. These results were obtained after 1000 initial cycles, and using 500 final cycles with the noise levels indicated in the table. A background intensity of 0.4 and an additional image intensity of 0.4 were used in each case. Noise was added to all the image pixels. For example, adding random values up to ± 0.04 to the image intensities results in a maximum noise level of 10%, as shown in Table I. No attempt was made to optimize the learning or other RNN parameters in these runs, the purpose being to study the performance with increasing noise, and as the number of Mnodes is increased.

The performance of the MH RNN is seen to decrease smoothly with increasing

Maximum noise		% correct		
Intensity	%	Number of Mnodes		
		4	6	8
0.02	5	99.7	100.0	100.0
0.04	10	98.0	100.0	100.0
0.08	20	90.5	99.3	99.2
0.12	30	82.7	94.3	96.0
0.16	40	71.9	86.0	90.4
0.20	50	64.1	76.9	83.8

Table I: Effect of increased noise and number of Mnodes on RNN identification of 4x6 images after MH training with fixed learning parameters

noise level, and to increase with the number of Mnodes used. Both results are expected, the latter because using more Mnodes increases the dimensionality of the space onto which the unique Rnode state is projected, which increases the separation of the classes.

The separation of the class vectors in the n_m space is affected by the values used for the learning rates, and the other RNN parameters. It may be expected that increasing the separation of the class vectors would improve the identification performance of an RNN. For a given number of Mnodes, however, this does not produce reliable improvement with noisy images; i.e. the class vectors can be separated somewhat by adjusting the parameters, but that does not always result in improved identification percentages.

For any given RNN configuration, the identification performance is affected by varying the following parameters: $\alpha, \beta, \gamma, s_r$, as well as the number of initial cycles, the exposure time, and the constant weight used for S-Rnode connections. An approximate optimum can be obtained by trial and error, but it is not very sensitive in a region of that optimum. For example, with no training at all, i.e. just using fixed initially random weights, a 100 Rnode system can give about 70% correct identification with 40% noise, which can be increased to about 81% by parameter adjustment.

Table II shows optimal identification performance of a 100 Rnode network with 40% noise, with increasing numbers of Mnodes. With more than 6 Mnodes, the effect of varying the learning parameters is minimal. More than 90% of the images can be successfully identified with a random initial network with fixed weights and offsets (no training) if more than 6 Mnodes are used. This appears to indicate that the RNN dynamics is unique for unique images even without adjusting the connection weights and node offsets, and leads to unique projections onto sufficient Mnodes. The MH training process only appears to be useful if a highly compressed representation of

Number of Mnodes	4	6	8	10	12	14	16
% correct (± 1)	81	92	93	94	95	96	97

Table II: Optimal RNN identification of 4x6 images with 40% maximum noise, after MH training

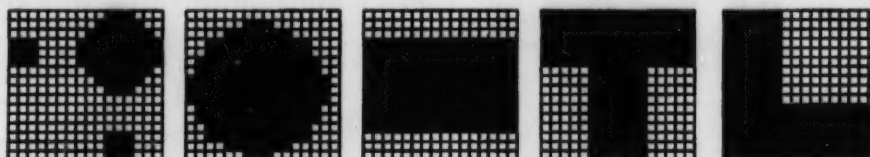


Figure 3.3: Five training images using 256 sensors (16x16 array)

the images is required, for then significant improvement in the identification rate can be achieved.

Much larger images have also been successfully identified. The set of five 16x16 images shown in Figure 3.3 was used (256 Snodes) as input to an RNN with 300 Rnodes (200 in the input region) and 4, 5 and 6 Mnodes. Each Snode had 10 output connections, the Rnodes and Mnodes had 20 and 10 input connections respectively. The results, after 1000 training cycles and 500 test (final) cycles, are given in Table III.

In this case a large reduction in the dimension of the vector required for identification (data compression) is achieved (approximately 50:1), and optimization of the RNN parameters leads to a correct identification rate of almost 100% with 5 Mnodes, even with up to 50% noise. Of course, this is only for a limited number of quite different images. It should also be noted that with the RNN used, there may be a loss of significant image information in going from the 256 Snodes to 200 Rnodes in the input region. This means that a maximum of 200 different combinations of the image data are generated, which may not be sufficient to allow the RNN to distinguish between more similar patterns.

3.3 Problems with the MH Technique

When an MH RNN is operating with a fixed image in the Snode array, the RNN state moves into a region of its space that is similar for similar images. A region of the state space could be defined by an n_m -dimensional radius from the points resulting from the MH training, which would define an acceptable degree of similarity for classification. If a new image were outside any of the defined regions, new classes could be created. With only MH learning, the RNN is therefore essentially being used as a clustering

Maximum noise		% correct		
Intensity	%	Number of Mnodes		
		4	5	6
0.04	10	100.0	100.0	100.0
0.08	20	99.7	100.0	100.0
0.12	30	98.0	100.0	100.0
0.16	40	95.8	99.5	99.8
0.20	50	92.8	98.6	98.2

Table III: RNN identification of 16x16 images after MH training

technique, which leads to the following questions:

- why use a neural network?
- why not just store the training set images as class vectors? and
- how is this technique related to machine intelligence?

It is true that there is data compression in the Mnodes used for identification, and there may be some form of feature extraction occurring in the Rnodes, although this has not been demonstrated here. However, we have also used the training set images themselves as class vectors, and without the use of the RNN at all it is possible to identify 100% of the images even with a maximum noise level of 50%, simply by calculating the n_s -dimensional distances and choosing the closest one. Of course, this requires calculating image intensity differences at every pixel, which is a lot of calculations for large images, but then so is the operation of a large RNN.

It is not clear how the convergent, clustering properties of the MH RNN are related to our concept of machine intelligence, for in the MH RNN system there is:

- no explicit memory of repeated images;
- no internal mechanism to indicate recognition; and
- no internal mechanism for association of different types of sensor data.

These deficiencies are addressed in Chapters 4 and 5.

UNCLASSIFIED

UNCLASSIFIED

DRES-SR-709

4. Explicit Memory (Resonant Feedback Learning)

By using a technique, such as modified Hebbian learning, that simply updates the connection weights and node offsets based on the strength of signals propagated through an RNN by an image that is repeatedly presented, it may be that some form of imprint of the image is recorded in the network. However, in our system, we have no evidence that there is explicit memory of the images seen, or that there is any way of regenerating an image internally in the RNN using only MH learning.

To store and regenerate explicit representations of images, we have used feedback signals derived from differences between the image presented in the Snodes and the resultant intensities generated in the Mnode array. In this case, the Mnode and Snode arrays must have the same dimensions.

Figure 4.1 shows the concept. Each sensor node, shown on the left, has a unique set of connections to Rnodes in the input region, and each Mnode has a unique set of inputs from Rnodes in the output (or *memory*) region of the RNN. The aim is to match an Mnode intensity with that in the corresponding Snode, by adjusting the input connection weights of the Rnodes that connect to that Mnode. The connections between the Rnodes and the Mnodes are themselves kept constant (± 1.0). The Mnode offsets are not changed in this learning scheme either; they are fixed at 0.0. Recall that in the simple MH learning scheme, the Mnode offsets can shift to ensure that they can generate signals other than 0 or 1. However, in the resonant feedback (RF) scheme, or any other explicit memory technique, Mnodes may be required to have outputs close to 0 or 1, and the *responsibility* for the values that are generated by the Mnodes is passed to the Rnodes in the memory region. Fig. 4.1 indicates that the strength of the feedback varies between 0.0, when the Snode and Mnode are exactly matched, and 1.0 when there is a strong mismatch.

We define a Gaussian resonance function, which can also be used as a measure of the correlation between the Snode and Mnode arrays, in the following way:

$$R = \frac{1}{n_s} \sum_i^{n_s} e^{-\lambda \Delta_i^2} \quad (4.1)$$

UNCLASSIFIED

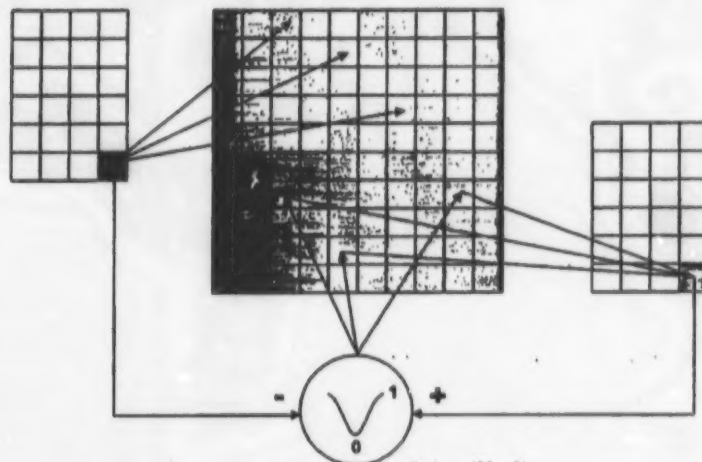


Figure 4.1: An MH-RNN with Resonant Feedback

where Δ_i is the difference between the output signals generated by the i th Mnode and the i th Snode, and λ is related to the full width at half maximum (FWHM) of the correlation function by:

$$\lambda = -4 \ln(0.5) / (\text{FWHM})^2 \quad (4.2)$$

The value for FWHM can be set in the RnnPr Parameters dialog (Fig. 2.4), in the edit box labelled **Resonant width**. This allows a user to set the sensitivity of the correlation. The resonant feedback curve shown in Fig. 4.1 is, of course, just $(1.0 - R)$, and R becomes 1.0 when the images are matched.

An Rnode in the memory region may be connected to many Mnodes. For any Rnode that *does* have output connections to Mnodes, we wish to change the input connection weights to that Rnode so that the correlation of the connected Mnodes is improved. We define a desired *direction* for the change in output signal strength of the n th Rnode by the following:

$$D_n = -\frac{1}{n_{mcons}} \sum_i^{n_{mcons}} W_{ni} \Delta_i \quad (4.3)$$

In Eq. 4.3, the number of connections to Mnodes from the n th Rnode is n_{mcons} , and in general this is different for each Rnode. W_{ni} is the connection weight from the n th Rnode to Mnode i , and this weight is either +1.0 or -1.0. D_n itself will have a value between +1.0 and -1.0.

D_n can be understood as the resultant of a sum of competing forces. To understand the negative sign in Eq. 4.3 consider the following: if a particular Mnode has a positive value for Δ , its output is too high, and if the weight connecting it to the Rnode is

UNCLASSIFIED

DRES-SR-709

-1.0 then it would be desirable for the output of that Rnode to increase, hence the negative sign; conversely, if the weight is +1.0 a decrease in Rnode output is desired. On the other hand, if Δ is negative, the Rnode output should be increased if the weight is +1.0 and decreased if the weight is -1.0, and again the negative sign assures this. The summation causes all the connected Mnodes to compete for a change in the output of this Rnode, and the magnitude of the mismatch Δ determines the relative strengths of the Mnodes in the competition. Of course, each Mnode is connected to many Rnodes, and though an Mnode may lose a competition for direction at a particular Rnode, all Mnodes can eventually be satisfied if there are sufficient unique sets of connections. It is clear that we can expect to be able to match a *single* sensor image because each Mnode has a unique set of Rnode connections. However, it is not assured that a set of many different sensor images can be matched and retained in the memory region, but we will show that this is possible using the weight and offset updating method described below.

A change, in the desired direction, of the output of the n th Rnode is achieved by adjusting the Rnode input connection weights. At cycle t , the change in the j th input connection weight is calculated in the following way:

$$\Delta W_{jn}^t = \rho D_n^t f_{C_{jn}}^{t-1} W_{jn}^t S_{jn} \quad (4.4)$$

where ρ is the RF learning rate, which can be adjusted in the dialog box shown in Fig. 2.4. The effect of the remaining terms in Eq. 4.4 is to link the magnitude of a weight change to the strength of the incoming signal f on the j th input connection (from the Rnode whose index is given by C_{jn}), and to the strength of the weight itself (W_{jn}). The sign term, S_{jn} , ensures that the direction of the weight change, an increase or decrease, will produce the desired change in the Rnode output. Equation 4.4 also ensures that the weights are bounded as the Snode and Mnode arrays converge, since $\{\Delta_i\} \rightarrow 0$, and no weight can change sign because the weight change is of the form:

$$W_{jn}^{t+1} = W_{jn}^t (1 \pm |\epsilon|) \quad (4.5)$$

and $|\epsilon| < 1$.

The Rnode offset changes are almost the same as in MH learning (Eq. 3.2), but with the inclusion of the resonance correlation term thus:

$$\Delta x_{on}^t = (1.0 - R)\beta(f_n^t - 0.5) \quad (4.6)$$

so that the offsets of Rnodes that are involved in explicit memory cease to shift as the images become matched.

Equations 4.1 to 4.6 define the technique that we have called Resonant Feedback (RF) learning. For all Rnodes that do not have output connections to Mnodes, the

MH learning rules are used; i.e. it is only in the memory region of the Rnodes, and only for Rnodes connected to Mnodes, that RF learning is used.

4.1 RF Results

To assess the effectiveness of an RF-RNN in memorizing a set of images, we use the following total correlation function:

$$C = \frac{1}{N} \sum_j^N R_j \quad (4.7)$$

where R_j is given by Eq. 4.1 for the j th image, and N is the number of images. If every image is accurately reproduced during training, C will approach 1.0. During training, image j is held in the Snode array for the exposure time, which we call e_t . After e_t cycles, R_j is calculated and the next image is presented for e_t cycles. C is calculated over each complete cycle of all images.

The following parameters can affect the memory process in the RF-RNN: the constant value used for the Snode to Rnode weights; the MH learning parameters, α, β, γ ; the Rnode sigmoid steepness s_r ; the number of Rnodes; the number of output connections for the Snodes; the number of input connections for the Rnodes and Mnodes; the exposure time, e_t ; and finally the RF learning rate, ρ . For a given stable RNN configuration, the last two parameters (e_t and ρ) may be optimized.

It might be expected that by increasing e_t we could improve the image correlations and by increasing ρ we could speed up the memorization process. However, if the exposure time is made very long (e.g. > 100 cycles), each image is eventually very well reproduced in the Mnode array, and $C \rightarrow 1.0$, but if e_t is very high only the last image seen is remembered, and after training is stopped only the last image can be reproduced. Also, if the RF learning rate ρ is increased, the training eventually becomes unstable.

The effect of making ρ too high is shown in Fig. 4.2, in which the smooth curve ($\rho = 0.02$) indicates stable memorization of all the images, and the curve for $\rho = 0.10$ indicates an unstable RNN system which eventually does not have any memory of any of the images. In both cases there is an initial rapid improvement in the total image correlation, from about 0.5 for the initially random Mnode array to greater than 0.7 in the first 5 cycles.

With the stable value of $\rho = 0.02$, and an exposure time of 10 cycles, it was possible to obtain a good correlation and memory of all the ten 4x6 test images using an RNN with 200 Rnodes, with 20 input connections for the Rnodes and Mnodes, and 20 output connections for the Snodes. The Rnode array had 100 nodes in each of the input and memory regions. After training for 2000 complete cycles of all ten 4x6 images, the RF

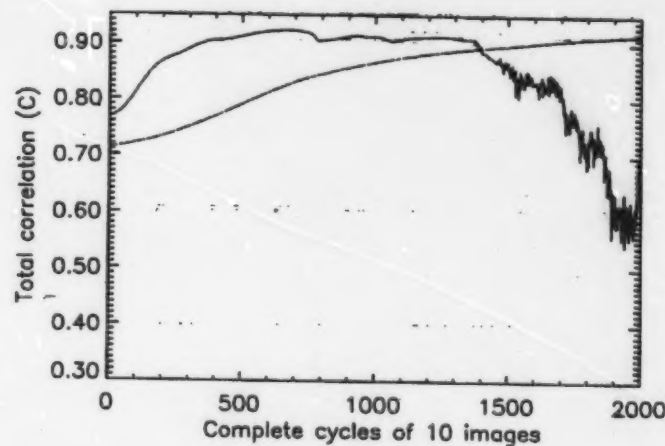


Figure 4.2: Total correlation of ten 4x6 images during RF training with $\rho = 0.02$ (smooth) and $\rho = 0.10$ (unstable)

learning was stopped; i.e. all input connection weights were fixed for those Rnodes in the memory region that were connected to Mnodes. MH learning was allowed to continue for the other Rnodes, however, during the recognition and classification process. Figure 4.3 shows two examples of the RNN output when previously seen images were presented. The Mnode array clearly gives a close reproduction of the remembered image. Because the purpose here was only to demonstrate that the RF training does work for multiple images, the RNN in this example was not fully optimized, only ρ and ϵ_t were varied for the fixed configuration. Even with this limited optimization, the correlation function values for the the ten individual images were all between 0.88 and 0.96.

If an image is presented which was *not* one of the training set, then there is no discernable reproduction of the input image if it is quite unlike any trained image. For example, we used three images representing the letters F, X and T as input, and obtained no recognizable reproduction of any of the input images. The correlation function values were between 0.57 and 0.74 for these images. These results show that the resonance correlation function R (Eq. 4.1) can provide a good indication of recognition.

The RF-RNN can also be used as a classifier, using the same process described for the MH-RNN in Chapter 3. The Mnode arrays resulting from the training cycles are stored as class vectors, and when an image is presented for identification the stored class vector that is nearest (Eq. 3.4) to the resulting Mnode array is taken as the identity of the input image. As with the MH-RNN, the RF-RNN is tolerant of noise in the input images. With 40% maximum noise, 98% of the ten 4x6 images were

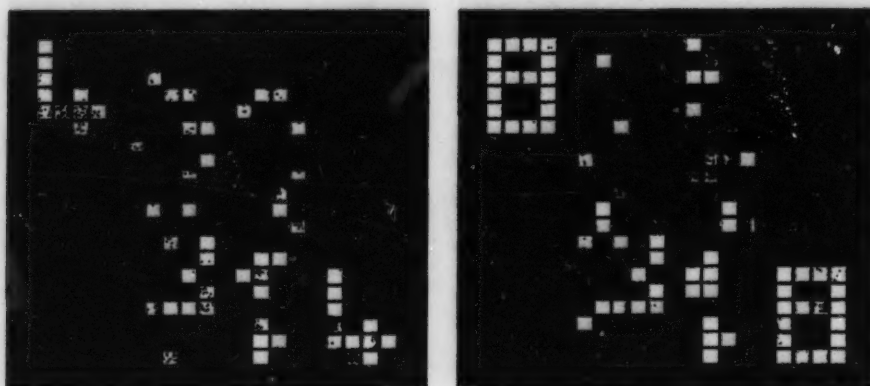
UNCLASSIFIED

Figure 4.3: Two examples of RNN output after RF training

correctly identified in 500 presentations of each image, and with 50% maximum noise 96% were correct. This is better than the MH results given in Tables I and II, but of course the RF-RNN does use more output nodes.

UNCLASSIFIED

DRES-SR-709

5. Associative Memory Design

We have demonstrated that a recurrent neural network operating with a form of Hebbian learning can retain an explicit memory of images that it has seen repeatedly if a feedback signal is used to update node input connection weights in a memory region of the network. The resonant correlation signal generated by the RF-RNN may also be used as a measure of recognition; i.e. in this system there is an internal mechanism for recognition, but as yet there is no internal process to generate *identification*. It is true that the user can determine that an image probably belongs to a particular class, but the *naming* of the classes must be done externally by the user.

In a self-learning system, the name or significance of an observed pattern must be derived from an association with another form of sensor image that repeatedly occurs simultaneously, thus establishing a *context* for the two images, neither of which has significance without the other during training. Associations between more than two sensor streams are possible in biological brains, but initially we will focus on designs that may be capable of autonomously forming associations between only two sensor arrays. The software implementation and testing of the RNN designs is under way at DRES.

Our goal is to build a system that not only has explicit memory of two forms of sensor images that it has seen, but also can form an association between them, so that when an image in only one of the sensor arrays is present it will automatically generate a close representation of the image in the other sensor array with which it was repeatedly associated during training.

The equal importance of both types of sensor image results in the symmetry found in the two designs presented here. In the diagrams for both cases we use the letter P to represent a *pattern*, and C to represent its code, which establishes the significance (name, classification, identity..) of the pattern. Of course, P and C are conceptually interchangeable.

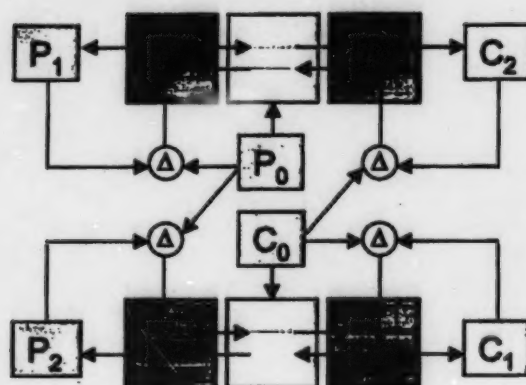


Figure 5.1: RF-RNN cross-linked associative memory design

5.1 Cross-Linked Structure

The first design for an RF-RNN associative memory is shown in Fig. 5.1. P_0 and C_0 represent different input sensor arrays that are connected to *different* input regions (shown in light grey) of an RNN. In the upper half of the diagram, a memory region (upper left) can store explicit versions (P_1) of the P_0 images that are seen repeatedly. Similarly, the lower right memory region stores versions (C_1) of C_0 . The symbol Δ indicates RF learning based on the differences between memory and sensor images. Explicit memory of P and C sensor images, and indications of recognition based on the Δ s, are thus able to operate independently. However, if images in P_0 and C_0 repeatedly occur *together*, then the upper right part of the P region of the RNN can also store a version (C_2) of C_0 , and at the same time the C region (lower left) can store a version (P_2) of P_0 . Associative memory is thus implemented by the *cross-linked* feedback indicated between the upper and lower halves of Fig. 5.1.

After training with repeated sets of pairs $\{P_{0i}, C_{0i}\}$ of images, it is anticipated that a presentation of an image P_{0i} , even in the absence of the corresponding C_{0i} image, will generate an output C_{2i} that is a close representation of C_{0i} . Similarly, C_{0i} alone may generate $P_{2i} \simeq P_{0i}$. In both cases an indication of recognition is also generated, in the resonances between P_{0i} and P_{1i} , and between C_{0i} and C_{1i} respectively.

Thus a presentation of P_{0i} , for example, can give indications of both *recognition* and *identification*. Recognition means: "I know I've seen this before because of the strong resonance correlation value", and identification means: "It's a C_{2i} ".

This design offers the possibility of entirely autonomous explicit memory, recognition, associative learning and identification for two sensor input arrays.

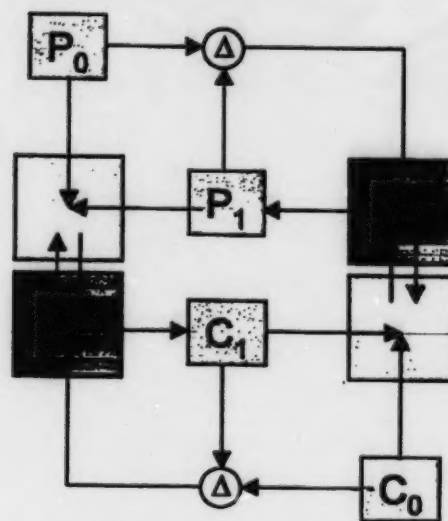


Figure 5.2: RF-RNN circular associative memory design

5.2 Circular Structure

A possible disadvantage of the cross-linked structure is that each learned image must be stored in two memory regions; e.g. $P_0 \rightarrow P_1$ and P_2 . The second design, shown in Fig. 5.2, is an attempt to generate association with only one storage region for each type of sensor input.

In the circular structure it is expected that memory of images can only occur effectively for repeated, simultaneous pairs. If images P_{0i} and C_{0i} are present simultaneously in the input sensor arrays for a long enough time, C_{1i} and P_{1i} will be generated and will contribute to the strength of the input images. This may require that the C_1 and C_0 arrays have the same sets of output connections; i.e. the i th node in C_1 will be connected to the same set of Rnodes as the i th node in C_0 . Similarly, the P arrays may have the same sets of output connections. In effect, the whole structure will then resonate during associative learning. If the C input is constantly changing, however, it will be difficult or impossible for a good representation of P_{0i} to form in P_1 because the input signals to the Rnodes in the memory region (upper right in Fig. 5.2) will not be stable. After training with a set of image pairs, when P_{0i} is presented, it should generate a representation C_{1i} (indicating an identification) and that in turn should generate a version of P_{1i} . If the image in C_0 does not correspond to the stored associated image C_{1i} , then the resonant signal Δ_c will be low. Whichever resonant signal is stronger, Δ_p or Δ_c , indicates which is the recognized input. If both are strong, then both are recognized. If one is below some threshold, that sensor input could be inhibited in order to strengthen the resonant identification process of the other.

UNCLASSIFIED

UNCLASSIFIED

DRES-SR-709

6. Discussion and Conclusions

The primary aim of this work was to determine if the same basic RNN structure and learning rules that were shown in previous work to be capable of generating goal-driven behaviour in a mobile machine could also be used to remember, recognize and classify patterns. If successful, it would then be possible to pursue the following longer term goals for RNN systems: a) to link motion with memory and multiple sensor input to provide adaptive control of a mobile machine that would improve its ability to navigate and reach goals; and b) to develop a more general, adaptive, autonomous image classifier than those available using standard statistical techniques.

The results in Chapter 3 show that the simple, biologically inspired Hebbian rule for altering the connection strengths, coupled with a mechanism for shifting the centre of activity of each neuron, does result in a system that may be used as a clustering classifier. Different sensor images drive the MH-RNN into unique regions of the Rnode space, and similar patterns cluster around similar points in this space. Conditioning the system by repeatedly presenting a set of images may be used to generate class vectors that can be stored and recalled to identify similar images that were not part of the training set. This classification method is quite tolerant of noise, and correct identification of greater than 90% of images can be expected in the presence of up to 50% maximum noise. The memory of images in this system is implicit, or indirect, because there does not seem to be any way of regenerating the actual images with which it was trained. Using the MH-RNN technique, it is possible to generate compressed forms of the input images that retain sufficient information for identification. The results obtained using 16x16 images in particular show significant data compression. It is also possible that this RNN system has automatic feature extraction capabilities, because the Rnodes in the input region may have input connections from unique combinations of sensor nodes; i.e. certain Rnodes will respond strongly to particular features in the images. Furthermore, the transfer of signals between the input and output regions of the RNN will generate unique combinations of these features so that Rnodes that are sensitive to complex feature relationships are possible. It may be possible to exploit this capability in future work.

For the storage and recall of explicit images, and for the ongoing development of automatic associative memory, it was necessary to augment the modified Hebbian learning with a form of feedback relating the images in the sensor and motor node arrays. Several different weight and offset updating schemes were tried before a successful implementation of direct memory, and the ability to explicitly regenerate multiple images from a training set, was achieved using the resonant feedback technique described in Chapter 4. The memory that results from this technique occurs relatively quickly. Even for the quite demanding task of remembering ten images each containing 24 channels of information (4x6), good storage can be achieved in about 1000 complete cycles, which take only about 200 seconds on a single CPU 300 MHz Pentium II computer. As a comparison, if one attempted to use the standard back-propagation neural network for this task it would certainly take tens of thousands of cycles, and may never converge satisfactorily.

By explicit memory, we mean the storage of sensor data in such a way that it can be recalled as it appeared, more or less, at the sensor during the memorization process. This does not mean that a memorized image is precisely stored pixel by pixel, but that a close reproduction of a memorized image is generated by the stimulus of sensor input. With resonant feedback learning, memory is encoded in the input connection weights of the Rnodes in the memory region of an RNN, and the reproduction of a remembered image is caused by an interaction throughout the entire RNN, from the sensor array to the memory display array. Thus the memory is *distributed*, and in the present system, a memory is only recalled by the presentation of an image that has been seen during training, or one that is quite similar. An image that has not been seen before, and is not like any seen, does not generate an indication of recognition.

The storage capacity of an RF-RNN will depend on the number of Rnodes and the number of their input connections. The exactness of the reproduction of remembered images will also depend on the number of input connections to the Mnodes and the number of output connections from the sensor nodes. These parameters are easily scalable in our C++ implementation of the RNN, so that larger images and bigger training sets are feasible. Other RNN parameters will affect the rate, stability and efficiency of learning, and these scalability issues must be investigated in future work.

Because of the ability to store images explicitly using RF learning, it has been possible to develop feasible designs for automatic associative memory systems, as described in Chapter 5. If successful, these self-learning systems will have potential application in many areas, including the following that are of immediate relevance to DRES R&D: multi-sensor data fusion; recognition and identification of objects in images, particularly air-ground targets and land mines in IR images; and adaptive, autonomous navigation of unmanned vehicles.

The Resonant Feedback learning method has the following advantages over the simple Modified Hebbian method and other classifier approaches that are based on clustering algorithms:

- images are remembered and can be recalled explicitly;
- there is an internal mechanism that indicates recognition;
- automatic association, and hence identification may be possible; and
- because the input region can maintain active MH learning, it may be possible to link goal-driven behaviour with associative memory, using feedback to sensors from the environment.

A system with a self-learning capability can also be taught, for example by a human teacher using repetition of significant relationships in two types of sensor inputs. In speech recognition for example, a user's sound patterns would be encoded as images that would be automatically associated with supplied words. There are many such potential applications.

The implementation and testing of the associative memory designs presented in Chapter 5 will provide a very interesting and promising basis for the future work to be carried out under the Technology Investment Fund project, 2FE20, to demonstrate self-organized, goal-driven adaptive learning in a simulated mobile vehicle.

UNCLASSIFIED

UNCLASSIFIED

DRES-SR-709

References

- [1] Bow, S-T., *Pattern Recognition, Application to Large Data-Set Problems*, Marcel-Dekker Inc., New York, 1984.
- [2] Kohonen, T, "Correlation Matrix Memories", *IEEE Trans. on Computers* C-21 (1972), p. 353.
- [3] Anderson, J.A., "A Simple Neural Network Generating an Interactive Memory", *Math. Biosci.* 14 (1972), p. 197.
- [4] Kohonen, T, "Self-organized Formation of Topologically Correct Feature Maps", *Biol. Cybern.* 43 (1982), p. 59.
- [5] Kohonen, T, *Self-Organization and Associative Memory*, Springer, Berlin, 1984.
- [6] Hopfield, J.J., "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons", *Proc Natl. Acad. Sci. USA*, 81 (1984), p. 3092.
- [7] Barton, S.A., "The Effect of Sensory Input on Trajectories Generated by Recurrent Neural Networks", DRES SR-589, 1993.
- [8] Barton, S.A., "Two Dimensional Movement Controlled by a Chaotic Neural Network", *Automatica*, 31 (1995), p. 1149.
- [9] Barton, S.A., "Structure and Convergence Properties of a Recurrent Neural Network", DRES SM-1489, 1996.
- [10] Kandel, E.R. and Hawkins, R.D., "The Biological Basis of Learning and Individuality", *Scientific American*, September, 1992.
- [11] *Neurocomputing, Foundations of Research*, Eds. Anderson J.A. and Rosenfeld, E., MIT Press, Cambridge MA, 1988.
- [12] *Neurocomputing 2, Directions for Research*, Eds. Anderson J.A. and Rosenfeld, E., MIT Press, Cambridge MA, 1990.

UNCLASSIFIED

- [13] *The Neurobiology of Neural Networks*, Ed. Gardner, D., MIT Press, Cambridge MA, 1993.
- [14] Hebb, D.O., *The Organization of Behaviour*, John Wiley and Sons, Inc., New York, 1949.

UNCLASSIFIEDDRES-SR-709
